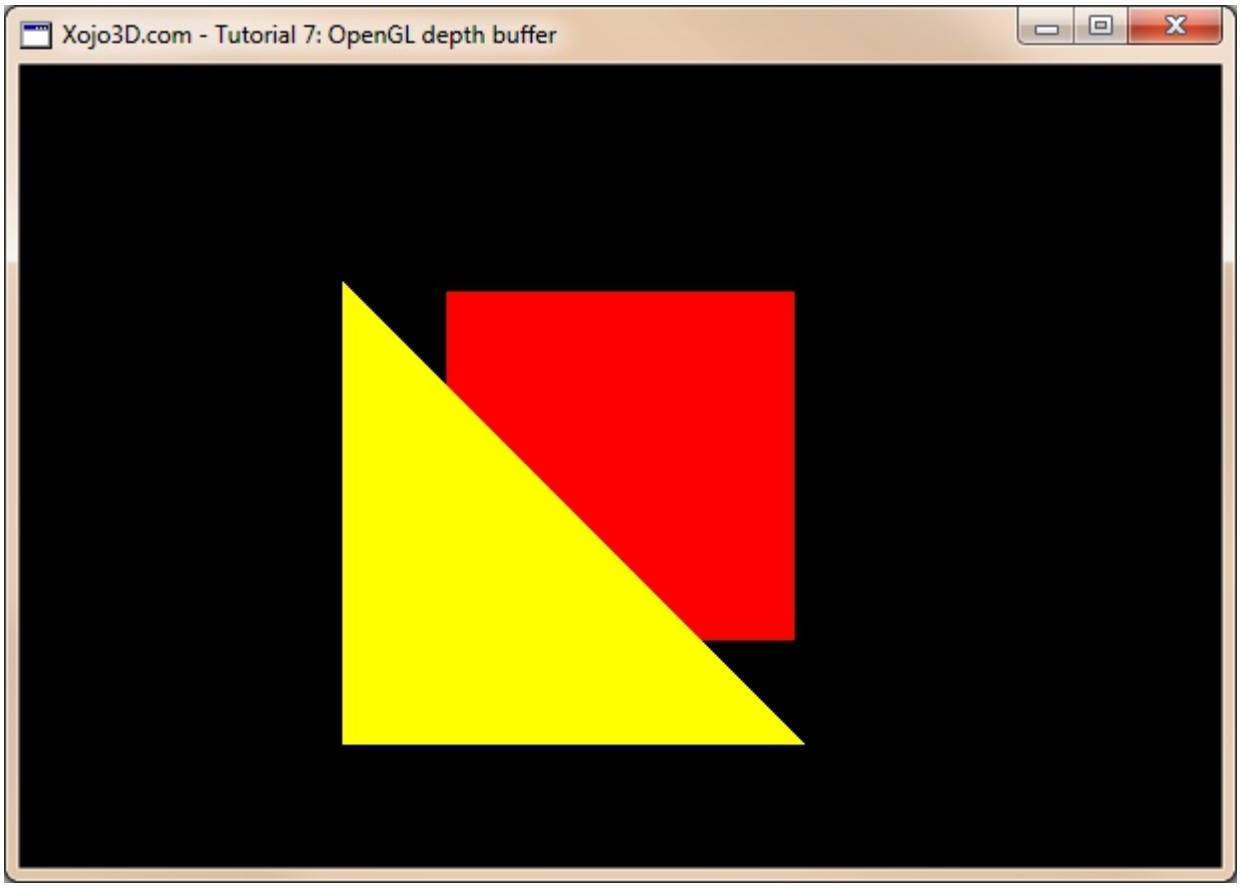
Tutorial 7: OpenGL depth buffer

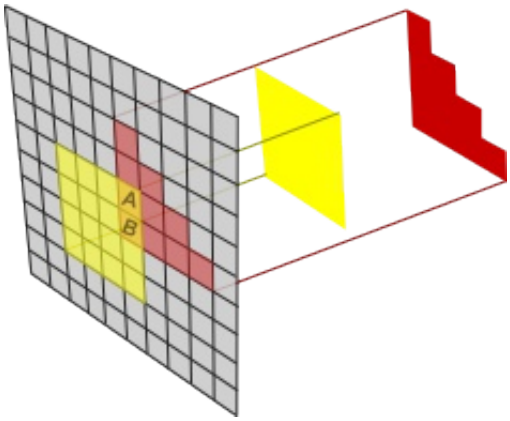
The OpenGL depth buffer prevents polygons at the back from being drawn over polygons in the front. Learn in this tutorial how to use the OpenGL depth buffer.



Theory

The OpenGL **depth buffer** stores depth information about each pixel drawn onto our surface. Since our surface is a 2-dimensional surface that only uses X and Y values, the depth buffer essentially provides a way to store the Z value of a pixel.

The information in the depth buffer is used to determine if the pixels of a polygon, is in front or behind the pixels that are already drawn onto the surface at the same location.



In the illustration above, the yellow square is drawn first. By the time that the red triangle is drawn, there is no way to determine if the pixels of the red triangle is in front or behind the yellow pixels on the surface at position A an B. This is where the depth buffer comes into play. Prior to drawing the red pixels, we first check the depth buffer at position A and B, to make sure the red pixels are in front of the yellow pixels. In the example above, the red pixels are behind the yellow pixels, and therefore not plotted onto the surface.

OpenGL has built-in depth testing, and we simply need to enable it to ensure that our polygons are drawn correctly.

Tutorial Steps

1. Open Xojo.
2. In the Project Chooser select Desktop.
3. Enter "Tutorial007" as the Application Name, and click OK.
4. Save your project.
5. Configure the following controls:

Control	Name	DoubleBuffer	Left	Top	Maximize Button
Window	SurfaceWindow	-	-	-	ON
OpenGLSurface	Surface	ON	0	0	-

6. Position and size *Surface* to fill the whole window, and set its locking to left, top, bottom and right.

7. Add the following code to the *SurfaceWindow.Open* event handler:

```
Self.MouseCursor = System.Cursors.StandardPointer
```

8. Add the following code to the *SurfaceWindow.Paint* event handler:

```
Surface.Render
```

9. Import the X3Core module, created in the previous tutorial.
   You can download the module from http://www.xojo3d.com/tutorials/tut007/x3core.zip.

10. Add the following code to the *Surface.Resized* event handler:

```
X3_SetPerspective Surface
```

11. Replace the code in the *X3Core.X3_Initialize* method with the following:

```
OpenGL.glEnable OpenGL.GL_DEPTH_TEST
OpenGL.glDepthMask OpenGL.GL_TRUE

OpenGL.glCullFace OpenGL.GL_BACK
OpenGL.glEnable OpenGL.GL_CULL_FACE
```

12. Add the following code to the *Surface.Open* event handler:

```
X3_Initialize
```

13. Add the following code to the Surface.Render event handler:

```
Dim i, j As Integer
Dim polygon() As X3Core.X3Polygon
Dim poly As X3Core.X3Polygon

poly = new X3Core.X3Polygon()
poly.FillColor = new X3Core.X3Color(1, 1, 0)
poly.Vertex.Append new X3Core.X3Vector(-1.2, 0.8, 0)
poly.Vertex.Append new X3Core.X3Vector(-1.2, -1.2, 0)
poly.Vertex.Append new X3Core.X3Vector(0.8, -1.2, 0)
polygon.Append poly

poly = new X3Core.X3Polygon()
poly.FillColor = new X3Core.X3Color(1, 0, -1)
poly.Vertex.Append new X3Core.X3Vector(-1, 1, -1)
poly.Vertex.Append new X3Core.X3Vector(1, -1, -1)
poly.Vertex.Append new X3Core.X3Vector(1, 1, -1)
polygon.Append poly

poly = new X3Core.X3Polygon()
poly.FillColor = new X3Core.X3Color(1, 0, -1)
poly.Vertex.Append new X3Core.X3Vector(-1, 1, -1)
poly.Vertex.Append new X3Core.X3Vector(-1, -1, -1)
poly.Vertex.Append new X3Core.X3Vector(1, -1, -1)
polygon.Append poly

OpenGL.glPushMatrix

OpenGL.glClearColor(0, 0, 0, 1)
OpenGL.glClear(OpenGL.GL_COLOR_BUFFER_BIT +
OpenGL.GL_DEPTH_BUFFER_BIT)

OpenGL.glTranslatef 0.0, 0.0, -3.0

OpenGL.glBegin OpenGL.GL_TRIANGLES

for i = 0 to polygon.Ubound

  poly = polygon(i)

// continue on next page
```

```
// continued from previous page

  if poly.FillColor <> nil then
    OpenGL.glColor3d(poly.FillColor.Red, poly.FillColor.Green,
poly.FillColor.Blue)
  else
    OpenGL.glColor3d(1, 1, 1)
  end if

  for j = 0 to poly.Vertex.Ubound
    OpenGL.glVertex3d poly.Vertex(j).X, poly.Vertex(j).Y,
poly.Vertex(j).Z
  next j

next i

OpenGL.glEnd

OpenGL.glPopMatrix
```

14. Save and run your project.

Analysis

X3Core.X3_Initialize:

```
OpenGL.glEnable OpenGL.GL_DEPTH_TEST
OpenGL.glDepthMask OpenGL.GL_TRUE

OpenGL.glCullFace OpenGL.GL_BACK
OpenGL.glEnable OpenGL.GL_CULL_FACE
```

In the initialization routine, we enable OpenGL depth testing with the first instruction. The glDepthMask call enables writing into the depth buffer. The last two lines enable back-face culling like explained in the previous tutorial.

Surface.Render:

```
Dim i, j As Integer
Dim polygon() As X3Core.X3Polygon
Dim poly As X3Core.X3Polygon

poly = new X3Core.X3Polygon()
poly.FillColor = new X3Core.X3Color(1, 1, 0)
poly.Vertex.Append new X3Core.X3Vector(-1.2, 0.8, 0)

// continue on next page
```

```
poly.Vertex.Append new X3Core.X3Vector(-1.2, -1.2, 0)
poly.Vertex.Append new X3Core.X3Vector(0.8, -1.2, 0)
polygon.Append poly

poly = new X3Core.X3Polygon()
poly.FillColor = new X3Core.X3Color(1, 0, -1)
poly.Vertex.Append new X3Core.X3Vector(-1, 1, -1)
poly.Vertex.Append new X3Core.X3Vector(1, -1, -1)
poly.Vertex.Append new X3Core.X3Vector(1, 1, -1)
polygon.Append poly

poly = new X3Core.X3Polygon()
poly.FillColor = new X3Core.X3Color(1, 0, -1)
poly.Vertex.Append new X3Core.X3Vector(-1, 1, -1)
poly.Vertex.Append new X3Core.X3Vector(-1, -1, -1)
poly.Vertex.Append new X3Core.X3Vector(1, -1, -1)
polygon.Append poly

OpenGL.glPushMatrix

OpenGL.glClearColor(0, 0, 0, 1)
OpenGL.glClear(OpenGL.GL_COLOR_BUFFER_BIT +
OpenGL.GL_DEPTH_BUFFER_BIT)

OpenGL.glTranslatef 0.0, 0.0, -3.0

OpenGL.glBegin OpenGL.GL_TRIANGLES

for i = 0 to polygon.Ubound

  poly = polygon(i)

  if poly.FillColor <> nil then
    OpenGL.glColor3d(poly.FillColor.Red, poly.FillColor.Green,
poly.FillColor.Blue)
  else
    OpenGL.glColor3d(1, 1, 1)
  end if

  for j = 0 to poly.Vertex.Ubound
    OpenGL.glVertex3d poly.Vertex(j).X, poly.Vertex(j).Y,
poly.Vertex(j).Z
  next j
```

When you study the rendering code, you will notice that we have two shapes made from three polygons. A yellow triangle closer to us and a red square in the distance.

It is important to note that we need to pass the GL_DEPTH_BUFFER_BIT constant to the glClear function, to clear the depth buffer before we start rendering our scene.

The important lesson to take from this code is that, even though we draw the red square after the yellow triangle, the red surface does not cover any part of the yellow surface, because the square is actually behind the triangle.

You can test the effect that depth testing has on rendering, by commenting out the first two lines in the X3_Initialize method. See how the red surface now covers the yellow surface.